Softwareparadigmen SS 2015, Übungsblatt P

Abgabe: 27. Mai 2015, bis 16:00 Uhr

1. Bilden Sie eine Zweiergruppe mit einem/einer weiteren LV-TeilnehmerIn
2. Erstellen Sie mittels TUGrazOnline ein SVN-Repository für ihre Gruppe nach dem Muster `https://svn.tugraz.at/svn/swp2015_<MatrikelNr1>_<MatrikelNr2>`.
   Beispiel für StudentInnen mit Matrikelnummern 1201234 und 1206789:
   `https://svn.tugraz.at/svn/swp2015_1201234_1206789`
3. Fügen Sie die Studienassistenten (Stephan Frühwirt, Claudio Kirchmair, Benedikt Maderbacher und Thorsten Ruprechter) als Reader hinzu („Bedienstete wählen“)
4. Beurteilt wird der letzte Commit vor der Deadline

Aufbau des Repository
Im Root-Ordner des Repository muss sich ein Textfile `group.txt` befinden, welches in genau zwei Zeilen die beiden Namen und Matrikelnummern der Gruppenmitglieder enthält. Formatvorlage (am besten kopieren):

```
Alice Musterfrau, 1201234
Bob Mustermann, 1206789
```

Auf der Lehrveranstaltungsseite finden Sie ein Framework Framework.zip entpacken Sie dieses in den Root-Ordner.
Ihre Lösungen werden automatisiert getestet, halten Sie sich daher bitte an folgende Ordnerstruktur, und stellen Sie sicher, dass ihr Projekt mit SBT compiliert, ausgeführt und getestet werden kann.
Vorlage für die Ordnerstruktur:

```
<Repository-URL>/group.txt
<Repository-URL>/SWPInterpreter/build.sbt
<Repository-URL>/SWPInterpreter/src/main/scala/AST.scala
<Repository-URL>/SWPInterpreter/src/main/scala/Interpreter.scala
<Repository-URL>/SWPInterpreter/src/main/scala/Parser.scala
<Repository-URL>/SWPInterpreter/src/main/scala/SWPInterpreter.scala
<Repository-URL>/SWPInterpreter/src/test/scala/SWPInterpreterTests.scala
```

Die Binärdaten in `SWPInterpreter/target` und `SWPInterpreter/project/target` sollen nicht mit eingecheckt werden.

Parser (15 P.)

Implementieren Sie einen Parser der folgende, an EXP angelehnte, Sprache parsen kann. Definieren Sie dazu einen Abstract Syntax Tree mittels case classes (AST.scala) und schreiben sie den dazu passenden Parser mit der Scala Parser Combinator Library (Parser.scala).
Die Grammatik ist gegeben als EBNF(Extended Backus–Naur Form)

```
program = funcDecls , expression;

funcDecls = "{" , [ { funcDecl , ";" } , funcDecl ] , "}"

funcDecl = id , "(" , [ { id , "," } , id ] , ")" , "=" , expression;

expression = cond | function | int | list | variable;

cond = "if" , predicate , "then" , expression , "else" , expression;

predicate = id , "?" , "(" , [ { expression , "," } , expression ] , ")";

function = id , "(" , [ { expression , "," } , expression ] , ")";

int = "0" | ( digitNoZero , {digit} ) | ( "-" , digitNoZero , {digit} );

list = "[" , [ { expression , "," } , expression ] , "]";

variable = id;

id = letter , { letter | letterC | digit };

digitNoZero = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;

digit = "0" | digitNoZero

letterC = "A" | "B" | "C" | "D" | "E" | "F" | "G"
       | "H" | "I" | "J" | "K" | "L" | "M" | "N"
       | "O" | "P" | "Q" | "R" | "S" | "T" | "U"
       | "V" | "W" | "X" | "Y" | "Z" ;

letter = "a" | "b" | "c" | "d" | "e" | "f" | "g"
       | "h" | "i" | "j" | "k" | "l" | "m" | "n"
       | "o" | "p" | "q" | "r" | "s" | "t" | "u"
       | "v" | "w" | "x" | "y" | "z" ;
```

Interpreter für eingebauten Funktionen (10 P.)

Schreiben Sie einen Interpreter der folgende Untermenge der Sprache(EXP mit Listen und Integer) beherrscht:

Eingebaute Funktionen

- plus(a,b) $\rightarrow a + b$
- minus(a,b) $\rightarrow a - b$
- mult(a,b) $\rightarrow a * b$
- div(a,b) $\rightarrow a/b$
- first(xs) $\rightarrow$ xs.head
- rest(xs) $\rightarrow$ xs.tail
- build(x,xs) $\rightarrow x :: xs$

Prädikate

- eq?(a,b) $\rightarrow a = b$
- lt?(a,b) $\rightarrow a < b$

Konditionale

- if p then a else b $\rightarrow a$ wenn $p = T$
- if p then a else b $\rightarrow b$ wenn $p = F$

Implementieren Sie dazu den Value AST sowie den dazu gehörigen Prettyprinter (AST.scala) und die interpret Funktion(Interpreter.scala). Achten Sie bei der Stringgenerierung darauf das die Vorgaben genau eingehalten werden, da wir diesen Output für unsere Tests verwenden!

User definierte Funktionen (15 P.)

Erweitern Sie den Interpreter um User definierte Funktionen. Die Funktionen sollen wie im Skriptum auf Seite 59 beschrieben mit einer call-by-value Semantik interpretiert werden.

Exceptions (10 P.)

Erweitern Sie den Interpreter um Exceptions. Dazu wird die Grammatik um folgende Expressions ergänzt:

```
expression = ... | throw | tryCatch

throw = "throw" , exceptionId;

tryCatch = "try" , expression , "catch" , "{" , [ { handler , ";" } , handler ] , "}";

handler = (exceptionId | "_") , ":" , expression;

exceptionId = letterC , { letter | letterC }
```

Semantik

Wenn eine throw Expression evaluiert wird, soll die weitere Evaluierung unterbrochen werden und die entsprechende Expression wird zurück geliefert. Eine Exception wird solange nach oben durch gereicht bis sie eine try catch expression erreicht, oder man ganz oben ankommt.
Im Falle eines try catch wird geprüft ob die Exception in der Liste der Handler vorkommt, wenn ja wird die zugehörige Expression ausgewertet. Underscore stellt ein catch all Statement da, wenn keine spezifischer Exceptionhandler vorhanden ist wird diese Expression ausgewertet. Sollte der richtige Exceptionhandler nicht vorhanden sein und auch kein catch all, wird die Exception weiter geworfen, muss also auf einer höheren Ebene behandelt werden.
Wenn ein Program mit einer unbehandelten Exception endet soll der entsprechende Outputtext generiert werden “Uncaught exception ID!”, wobei ID dem Exception Identifier entspricht.
Zusätzlich zu den User definierten Exceptions sollen auch die eingebauten Funktionen und Prädikate um entsprechende Exceptions ergänzt werden:

- TypeMismatch : Diese Exception wird immer ausgelöst wenn Funktionen/Prädikate auf inkompatiblen Typen aufgerufen werden. z.B. first(1), eq?(0, []), plus([1],[2])
- DivByZero : Wenn durch 0 dividiert werden soll. z.B. div(7,0)
- EmptyList : Wenn first oder rest auf eine leere Liste angewandt wird. z.B. fist([]), rest([])

Framework

Auf der Homepage finden Sie ein SBT Project SWPInterperter. In diesem ist eine `build.sbt` Datei sowie unter `src/main/scala` ein Grundgerüst für die Aufgabe enthalten.

Um das Projekt zu starten müssen Sie zuerst SBT installieren, dann können Sie die SBT Konsole starten indem sie eine Komandozeile im Projektverzeichnis(Das Verzeichnis in dem sich die build.sbt befindet) öffnen und `sbt` eingeben.

Zum compilieren geben Sie `compile` ein. Das erste Compilieren kann etwas länger dauern da, zuerst Scala und die notwendigen Libraries heruntergeladen und installiert werden müssen.

Um das Programm in test.exp zu interpretieren geben sie `run test.exp` ein.

Tests

In der Datei `src/test/scala/SWPInterpreterTests` finden sie einige Tests um ihr Program zu testen. Diese Tests decken nicht alle Funktionalitäten ab sondern sollen zur Orientierung dienen. Nutzten Sie diese Tests auch um zu überprüfen das ihre Ausgabe Formatierung korrekt ist.
Sie können auch weitere Tests zu SWPInterpreterTests hinzu fügen um ihre Implementierung zu testen, diese Tests können sie auch gerne über die Newsgroup, oder auf anderem Weg, mit anderen Studierenden teilen.

Tests ausführen

Die Test können einfach in der SBT Konsole ausgeführt werden:

```
> test
[info] SWPInterpreterTests:
[info] - Parser short program
[info] - Parser defect program
[info] - Interpreter program with only build in functions
[info] - Interpreter short program
[info] - DivByZero Exception
[info] - TypeMismatch Exception
[info] - Custom Exception
[info] Run completed in 182 milliseconds.
[info] Total number of tests run: 7
[info] Suites: completed 1, aborted 0
[info] Tests: succeeded 7, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[success] Total time: 0 s, completed Apr 29, 2015 1:24:33 PM
```

Viel Erfolg!